Modeling Constrained Preemption Dynamics Of Transient Cloud Servers

ABSTRACT

Scientific computing applications are being increasingly deployed on cloud computing platforms. Transient servers can be used to lower the costs of running applications on the cloud. However, the frequent preemptions and resource heterogeneity of these transient servers introduces many challenges in their effective and efficient use. In this paper, we develop techniques for modeling and mitigating preemptions of transient servers, and present SciSpot, a software framework that enables low-cost scientific computing on the cloud. SciSpot deploys applications on Google Cloud preemptible virtual machines, and introduces the first empirical and analytical model for their preemptions.

SciSpot's design is guided by our observation that many scientific computing applications (such as simulations) are deployed as "bag" of jobs, which represent multiple instantiations of the same computation with different physical and computational parameters. For a bag of jobs, SciSpot finds the optimal transient server on-the-fly, by taking into account the price, performance, and preemption rates of different servers. SciSpot reduces costs by 5× compared to conventional cloud deployments, and reduces makespans by up to 10× compared to conventional high performance computing clusters.

1 INTRODUCTION

Transient cloud computing is an emerging and popular resource allocation model used by all major cloud providers, and allows unused capacity to be offered at low costs as preemptible virtual machines. Transient VMs can be unilaterally revoked and preempted by the cloud provider, and applications running inside them face fail-stop failures. Due to their volatile nature, transient VMs are offered at steeply discounted rates. Amazon EC2 spot instances [**?**], Google Cloud Preemptible VMs [**?**], and Azure Batch VMs [**?**], are all examples of transient VMs, and are offered at discounts ranging from 50 to 90% compared to conventional, non-preemptible "on-demand" VMs.

To expand the usability and appeal of transient VMs, many systems and techniques have been proposed that seek to ameliorate the effects of preemptions and reduce the computing costs of applications. Fault-tolerance mechanisms, resource management policies, and cost optimization techniques have been proposed for a wide range of applications—ranging from interactive web services, distributed data processing, parallel computing, etc. These techniques have been shown to minimize the performance-degradation and downtimes due to preemptions, and reduce computing costs by up to 90%.

However, the success of these techniques depends on probabilistic estimates of when and how frequently preemptions occur. For instance, many fault-tolerance and resource optimization policies are parametrized by the mean time to failure (MTTF) of the transient VMs. For example, periodic checkpointing is a common technique for reducing the work lost due to preemptions, and the "optimal" checkpointing frequency that minimizes the total expected running time of a job depends on the MTTF of the VMs.

Past work on transient computing has focused on Amazon EC2's spot instances, whose preemption characteristics are determined by dynamic prices (which are in turn set using a continuous second-price auction). Transiency-mitigation techniques such as VM migration [**?**], checkpointing [**? ?**], diversification [**?**], *all* use price-signals to model the availability and preemption rates of spot instances. However, these pricing-based models are not generalizable to other transient VMs having a flat price (such as Google's or Azure's offerings). In these cases, the lack of information about preemption characteristics precludes most failure modeling and transient computing optimizations.

To address this gap, we seek to understand the preemption characteristics of Google's Preemptible VMs, whose distinguishing characteristic is that they have a *maximum lifetime of 24 hours*. We conduct a large empirical study of over 1,500 preemptions of Google Preemptible VMs, and develop an analytical probability model of preemptions. We find that the temporal constraint is a radical departure from pricing-based preemptions, and presents fundamental challenges in preemption modeling and effective use of Preemptible VMs.

Due to the temporal preemption constraint, classical models that form the basis of preemption modeling and policies, such as memoryless exponential failure rates, are not applicable. We find that preemption rates are *not* uniform, but bathtub shaped with multiple distinct temporal phases, and are incapable of being modeled by existing bathtub distributions such as Weibull. We capture these characteristics by developing a new probability model. Our model uses reliability theory principles to capture the 24-hour lifetime of VMs, and generalizes to VMs of different resource capacities, geographical regions, and across different temporal domains. To the best of our knowledge, this is the *first* work on constrained preemption modeling. Our investigation also points to a new, surprising connection to statistical mechanics, which can lead to new insights for modeling temporally constrained events.

We show the applicability and effectiveness of our model by developing optimized policies for job scheduling, checkpointing, and server selection. These policies are fundamentally dependent on empirical and analytical insights from our model such as different time-dependent failure rates of different types of VMs. These optimized policies are a building block for transient computing systems and reducing the performance degradation and costs of preemptible VMs. We implement and evaluate these policies as part of a new software framework for running scientific computing applications, called SciSpot.

SciSpot abstracts typical scientific computing workloads and workflows into a new unit of execution, which we call as a "bag of jobs". These bags of jobs, ubiquitous in scientific computing,

, ,
.

represent multiple instantiations of the same application launched with possibly different physical and computational parameters. The bag of jobs abstraction permits efficient implementation of our optimized policies, and allows SciSpot to lower the costs and barriers of transient VMs for scientific computing applications.

Towards our goal of developing a better understanding of constrained preemptions, we make the following contributions:

(1) We develop a probability model of constrained preemptions based on a large-scale, first-of-its-kind empirical study of lifetimes of Google Preemptible VMs for understanding and characterizing their preemption dynamics. Our model captures the key effects resulting from the 24 hour lifetime constraint associated with these VMs, and we analyze it through the lens of reliability theory and statistical mechanics.
(2) We develop optimized policies for job scheduling, periodic checkpointing, and VM selection, that minimize the total time and cost of running applications. These policies are based on our preemption models, and reduce job running times by up to 40% compared to existing preemption models used for transient VMs.
(3) We implement all our policies as part of a new framework, SciSpot, and evaluate the cost and performance of different representative scientific computing applications on the Google Cloud Platform. Compared to conventional cloud deployments, SciSpot can reduce costs of running bags of jobs by more than 5×, and when compared to dedicated HPC clusters, it can reduce the total turnaround time by up to an order of magnitude.

2 BACKGROUND

We now give an overview of transient cloud computing, and motivate the need for the bag of jobs abstraction in scientific computing workflows.

2.1 Transient Cloud Computing

Infrastructure as a service (IaaS) clouds such as Amazon EC2, Google Public Cloud, Microsoft Azure, etc., typically provide computational resources in the form of virtual machines (VMs), on which users can deploy their applications. Conventionally, these VMs are leased on an "on-demand" basis: cloud customers can start up a VM when needed, and the cloud platform provisions and runs these VMs until they are shut-down by the customer. Cloud workloads, and hence the utilization of cloud platforms, shows large temporal variation. To satisfy user demand, cloud capacity is typically provisioned for the *peak* load, and thus the average utilization tends to be low, of the order of 25% [**?** **?**].

To increase their overall utilization, large cloud operators have begun to offer their surplus resources as low-cost servers with *transient* availability, which can be preempted by the cloud operator at any time (after a small advance warning). These preemptible servers, such as Amazon Spot instances [**?**], Google Preemptible VMs [**?**], and Azure batch VMs [**?**], have become popular in recent years due to their discounted prices, which can be 7-10× lower than conventional non-preemptible servers. Due to their popularity among users, smaller cloud providers such as Packet [**?**] and Alibaba [**?**] have also started offering transient cloud servers.

However, effective use of transient servers is challenging for applications because of their uncertain availability [**?** **?**]. Preemptions are akin to fail-stop failures, and result in loss of the application's memory and disk state, leading to downtimes for interactive applications such as web services, and poor throughput for long-running batch-computing applications. Consequently, researchers have explored fault-tolerance techniques such as checkpointing [**?** **?** **?**] and resource management techniques [**?**] to ameliorate the effects of preemptions. The effect of preemptions is dependent on a combination of application resource and fault model, and mitigating preemptions for different applications remains an active research area [**?**].

2.2 Preemption Modeling of Transient VMs

Past work on transient computing has almost exclusively focused on the Amazon EC2 spot market, which has a radically different preemption model and dynamics compared to other transient cloud servers such as those offered by Google Cloud and Azure.

Launched in 2009, Amazon's EC2 spot instances are the first example of transient cloud servers. The preemptions of EC2 spot instances are based on their *price*, which is dynamically adjusted based on the supply and demand of cloud resources. Spot prices are based on a continuous second-price auction, and if the spot price increases above a pre-specified maximum-price, then the server is preempted [**?**].

Thus, the time-series of these spot prices can be used for understanding preemption characteristics such as the frequency of preemptions and the "Mean Time To Failure" (MTTF) of the spot instances. Publicly available[1] historical spot prices have been used to characterize and model spot instance preemptions [**?** **?**]. For example, past work has analyzed spot prices and shown that the MTTFs of spot instances of different hardware configurations and geographical zones range from a few hours to a few days [**?** **?** **?**]. (Prateek: sigcomm paper)

However the price-based approach for inferring preemption characteristics is of limited utility. Other transient VMs such as those offered by Google and Azure have *flat* pricing, and price-based techniques for preemption modeling and cost optimization are not applicable. Furthermore, price-based techniques may be of limited value after Amazon's recent change where preemptions were decoupled with pricing. To address this shortcomings, we develop *empirical* preemptions models for transient VMs. In particular, we focus on Google Preemptible VMs, whose preemption properties have been unknown so far. Additionally, their unique fixed lifetime leads to new challenges for conventional reliability theory models, and requires new modeling techniques, which we develop later in Section **??**.

2.3 Scientific Simulation Workloads

(Prateek: Not sure. Probably something on scientific workloads though?)

Large computational requirements of scientific simulations make them ideal for low-cost transient servers.

The typical workflow associated with most scientific computing applications, often involves evaluating a computational model across a wide range of physical and computational parameters. For instance, constructing and calibrating a molecular dynamics application (such as Nanoconfinement [**?**]), usually involves running

[1] Amazon posts Spot prices of 3 months, and researchers have been collecting these prices since 2010 [**?**].

a simulation with different physical parameters such as characteristic sizes and interaction potentials, as well as computational parameters such as simulation timesteps. Each of these parameters can take a wide range of values, resulting in a large number of combinations which must be evaluated by invoking the application multiple times (also known as a parameter sweep). Since each computational job explores a single combination of parameters, this results in executing a "bag of jobs", with each job in the bag running the same application, but with possibly different parameters.

The bag of jobs execution model is pervasive in scientific computing and applicable in many contexts. In addition to exploratory parameter sweeps, bags of jobs also result from running the application a large number of times to account for the model or computational stochasticity, and can be used to obtain tighter confidence intervals. Increasingly, bags of jobs also arise in the emerging research that combines statistical machine learning (ML) techniques and scientific simulations [**? ? ?**]. For instance, large bags of jobs are run to provide the necessary training and testing data for learning statistical models (such as neural networks) that are then used to improve the efficacy of the simulations [**?**].

Bags of jobs are analogous to, but distinct from the bag of *tasks* design [**?**], where applications are decomposed into independent processes to enable flexible task scheduling. In contrast, the bags of jobs abstraction is independent of the application design—a bag of jobs may consist of a synchronous MPI program that is not amenable to flexible scheduling.

The bag of jobs execution model has multiple characteristics, that give rise to unique challenges and opportunities when deploying them on transient cloud servers. Since bags of jobs require a large amount of computing resources, deploying them on the cloud can result in high overall costs, thus requiring policies for minimizing the cost and overall running time. The similarity in execution characteristics of jobs (such as their running times and parallel speedup) allows for bag-wide optimizations (Figure **??**). And last, treating entire bags of jobs as an execution unit, instead of individual jobs, can allow us to use partial redundancy between jobs and reduce the fault-tolerance overhead to mitigate transient server preemptions.

3 CONSTRAINED PREEMPTIONS OF GOOGLE PREEMPTIBLE VMS

To measure and improve the performance of applications running on transient cloud servers, it is critical to understand the nature and dynamics of their preemptions. In this section, we present empirical analysis of preemptions of Google Preemptible VMs, and develop new probabilistic models of their preemption rates.

3.1 Empirical Study Of Preemptions

To understand the nature of temporally constrained preemptions, we conducted the first empirical study of Google's Preemptible VMs, that have a fixed price and a maximum 24 hour lifetime. Our empirical study is necesitated by the fact that the cloud operator (Google) does not disclose any other information about the preemption rates, and thus relatively little is known about the preemptions (and hence the performance) of these VMs.

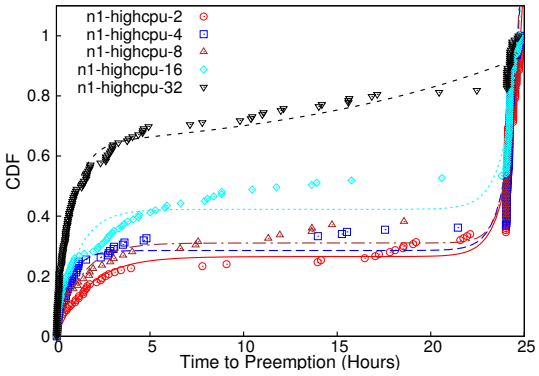

Figure 1: Preemption characteristics of different VM types. Larger VMs are more likely to be preempted.

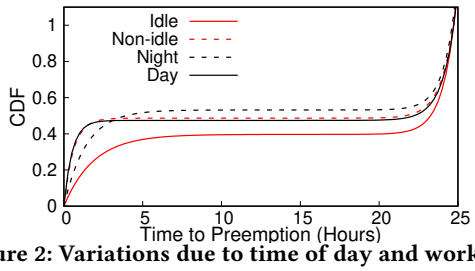

Figure 2: Variations due to time of day and workload.

We launched more than 1,500 Google Preemptible VMs of different types over a two month period (Feb–April 2019), and measured their time to preemption (i.e., their useful lifetime).[2]

A sample of over 100 such preemption events are shown in Figure **??**, which shows cumulative distribution function (CDF) of the VM lifetimes of the `n1-highcpu-16` VM in the `us-east1-b` zone. Note that the cloud operator (Google) caps the *maximum* lifetime of the VM to 24 hours, and all the VMs are preempted before that limit.

Observation 1: *The lifetimes of VMs are not uniformly distributed, but have three distinct phases.*

In the first (initial) phase, characterized by VM lifetime $t \in [0, 3]$ hours, we observe that many VMs are quickly preempted after they are launched, and thus have a steep rate of failure (derivative of the CDF) initially. In the second phase, VMs that survive past 3 hours enjoy a relatively low preemption rate over a relatively broad range of lifetime (characterized by the slowly rising CDF in Figure **??**). The third and final phase exhibits a steep increase in the number of preemptions as the preemption deadline of 24 hours approaches. The overall rate of preemptions is "bathtub" shaped as shown in the inset of Figure **??**.

Observation 2: *The preemption behavior, imposed by the constraint of the small 24 hour lifetime, is substantially different from conventional failure characteristics of hardware components and even EC2 spot instances.*

In "classical" reliability analysis, the rate of failure usually follows an exponential distribution $f(t) = \lambda e^{-\lambda t}$, where $\lambda = 1/\text{MTTF}$. Figure **??** shows the CDF ($= 1 - e^{-\lambda t}$) of the exponential distribution when fitted to the observed preemption data, by finding the distribution parameter λ that minimizes the least squares error. The

[2]We will release the complete preemption dataset for further analysis.

classic exponential distribution is unable to model the observed preemption behavior because it assumes that the rate of preemptions is independent of the lifetime of the VMs, i.e., the preemptions are *memoryless*. This assumption breaks down when there is a fixed upper bound on the lifetime, as is the case for Google Preemptible VMs, and the conventional approach becomes insufficient to model this constrained preemption dynamics.

Observation 3: *The bathtub shaped preemption behavior is a general, universal characteristic of Preemptible VMs.*

In general, the preemption dynamics of a VM is determined by the supply and demand of VMs of that *particular* type. Thus, our empirical study looked at preemptions of VMs of different sizes, in different geographical zones, at different times of the day, and running different workloads (Figure **YYY**). In all cases, we find that the preemption behavior is bathtub shaped and has the three preemption phases. We argue that this is not a coincindence, but may be a result of practical and fundamental outcomes of cluster management policies.

While the actual specific preemption policy is up to the cloud operator, we will show that the bathtub behavior has benefits for applications. For applications that do not incorporate explicit fault-tolerance (such as checkpointing), early preemptions result in less wasted work than if the preemptions were uniformly distributed over the 24 hour interval. Furthermore, the low rate of preemptions in the middle periods allows jobs that are smaller than 24 hours to finish execution with only a low probability of failure, once they survive the initial preemption phase. We compare job performance with bathtub shaped preemptions vs. uniform preemption rates in Section Evaluation.

In addition to being beneficial to applications, we also conjecture that the bathtub behavior is a *fundamental* characteristic of any system where events are randomly distributed in a finite interval. Intriguingly, we can analyze such temporally constrained preemptions through a statistical mechanics framework, and we elaborate on this connection later in Section **??**.

Observation 4: *Larger VMs have a higher rate of preemptions.*

Figure 1 shows the preemption data from five different types of VMs in the Google Cloud `n1-highcpu-{2,4,8,16,32}`, where the number indicates the number of CPUs. All VMs are running in the `us-central1-c` zone. We see that the larger VMs (16 and 32 CPUs) have a higher probability of preemptions compared to the smaller VMs. While this could be simply due to higher demand for larger VMs, it can also be explained from a cluster management perspective. Larger VMs require more computational resources (such as CPU and memory), and when the supply of resources is low, the cloud operator can quickly reclaim a large amount of resources by preempting larger VMs. This observed behavior aligns with the guidelines for using preemptible VMs that suggests the use of smaller VMs when possible [**?**].

Observation 5: *Preemptions exhibit diurnal variations, and are also affected by the workload inside the VM.*

From Figure 2, we can see that VMs have a slightly longer lifetime during the night (8 PM to 8 AM) than during the day. This is expected because fundamentally, the preemption rates are higher during periods of higer demand.

We also notice that completely idle VMs have longer lifetimes than VMs running some workload. Presumably, this could be a result of the lower resource utilization of idle VMs being more amenable to resource overcommitment, and result in lower preemptions.

3.2 Failure Probability Model

We now develop an analytical probability model for preemption dynamics that is faithful to the empirically observed data and provides a basis for developing running-time and cost-minimizing optimizations. Modeling temporally constrained preemptions raises many challenges for existing preemption models that have been used for other transient servers such as EC2 spot instances. We first discuss why existing approaches to preemption modeling are not adequate, and then present our closed-form probability model.

3.2.1 Inadequacy of existing failure distributions. Spot instance preemptions have been modeled using exponential distribution [**? ? ?**], which is the default in most reliability theory applications. However, the strict 24 hour constraint and the distinct preemption phases are not compatible with the memoryless properties of the exponential distribution. To describe failures (preemptions) that are not memoryless (i.e., increasing or decreasing failure rate over time), the classic Weibull distribution with CDF $F(t) = 1 - e^{-(\lambda t)^k}$ is often employed. However, the Weibull distribution is also unable to fit the empirical data (Figure **??**) and especially unable to model the sharp increase in preemptions near the 24 hour deadline.

For constrained preemptions, the increase in failure rate as modeled by the Weibull distribution is not high enough. Other distributions, such as Gompertz-Makeham, have also been used for modeling bathtub behavior, especially for actuarial use-cases. The key idea is to incorporate an exponential aging process, which is used to model human mortality. The CDF of the Gompertz-Makeham distribution is given by $F(t) = 1 - \exp\left(-\lambda t - \frac{\alpha}{\beta}(e^{\beta t} - 1)\right)$ and is fitted to the data in Figure XXX, and is also unable to provide a good model for the observed preemption data.

The non-trivial bathtub-shaped failure rate of Google preemptible VMs (Figure **??**) requires models that capture the sudden onset of the rise in preemptions near the deadline. From an application and transiency policy perspective, the preemption model must provide insights about the phase transitions, so that the application can adapt to the sharp differences in preemption rates. For example, the preemption model should be able to warn applications about impending deadline, which existing failure distributions cannot account for. Thus, not only is it important to minimize the total distribution fitting error, it is also important to capture the changes in phase. However, as we can see from the **QQ plots**, existing distributions are unable to capture the effects of the deadline and the phases of the preemptions, and a new modeling approach is needed, which we develop next.

3.2.2 Multiple failure rate model. Our failure probability model seeks to address the drawbacks of existing reliability theory models for modeling constrained preemptions. The key assumption underlying our model is the presence of two distinct failure processes. The first process dominates over the initial temporal phase and yields the classic exponential distribution that captures the high rate of early preemptions. The second process dominates over the final phase near the 24 hour maximum VM lifetime and is assumed

to be characterized by an exponential term that captures the sharp rise in preemptions that results from this constrained lifetime.

Based on these observations, we propose the following general form for the CDF:

$$\boxed{\mathscr{F}(t) = A\left(1 - e^{-\frac{t}{\tau_1}} + e^{\frac{t-b}{\tau_2}}\right)} \quad (1)$$

where t is the time to preemption, $1/\tau_1$ is the rate of preemptions in the initial phase, $1/\tau_2$ is the rate of preemptions in the final phase, b denotes the time that characterizes "activation" of the final phase where preemptions occur at a very high rate, and A is a scaling constant. In practice, typical fit values yield $b \approx 24$ hours, and $\tau_2 \approx 1$ hour, which ensures that our proposed distribution meets the initial condition $\mathscr{F}(0) \approx 0$.

For most of its life, a VM sees failures according to the classic exponential distribution with a rate of failure equal to $1/\tau_1$ – this behavior is captured by the $1 - e^{-t/\tau_1}$ term in Equation 1. As VMs get closer to their maximum lifetime imposed by the cloud operator, they are reclaimed (i.e., preempted) at a high rate $1/\tau_2$, which is captured by the second exponential term, $e^{(t-b)/\tau_2}$ of Equation 1. Shifting the argument (t) of this term by b ensures that the exponential reclamation is only applicable near the end of the VM's maximum lifetime and does not dominate over the entire temporal range.

The analytical model and the associated distribution function $\mathscr{F}$ introduced above provides a much better fit to the empirical data (Figure **??**) and captures the different phases of the preemption dynamics through parameters τ_1, τ_2, b, and A. These parameters can be obtained for a given empirical CDF using least squares function fitting methods. The failure or preemption rate can be derived from this CDF as:

$$f(t) = \frac{d\mathscr{F}(t)}{dt} = A\left(\frac{1}{\tau_1}e^{-t/\tau_1} + \frac{1}{\tau_2}e^{\frac{t-b}{\tau_2}}\right). \quad (2)$$

$p(t)$ vs. t yields a bathtub type failure rate function for the associated fit parameters (inset of Figure **??**).

In the absence of any prior work on constrained preemption dynamics, our aim is to provide an interpretable model with a minimal number of parameters, that provides a sufficiently accurate characterization of observed preemptions data. Further generalization of this model to include more failure processes would introduce more parameters and reduce the generalization power. Exploring other approaches of modeling bathtub-type failure rates (e.g., exponential Weibull distributions) [**? ?**] is part of our future work.

3.2.3 Reliability Analysis. We now analyze and place our model in a reliability theory framework.

Expected Lifetime: Our analytical model also helps crystallize the differences in VM preemption dynamics, by allowing us to easily calculate their expected lifetime. More formally, we define the expected lifetime of a VM of type i, as:

$$E[L_i] = \int_0^{24} t f_i(t)\, dt = -A(t+\tau_1)e^{-t/\tau_1} + A(t-\tau_2)e^{\frac{t-b}{\tau_2}}\Big|_0^{24} \quad (3)$$

where $f_i(t)$ is the rate of preemptions of VMs of type i (Equation 2).

This expected lifetime can be used in lieu of MTTF, for policies and applications that require a "coarse-grained" comparison of the preemption rates of servers of different types, which is a key requirement in cost-minimizing server selection, which we develop later in Section.

Hazard Rate: We now compute the hazard rate of our model, which is defined as $\lambda(t) = \frac{S(t)}{f(t)}$, where $S(t) = 1 - F(t)$ is the survival function. The hazard rate also related to the CDF as follows: $1 - F(t) = \exp \int_0^t -\lambda(x)\, dx$.

The hazard rate governs the dynamics of the failure processes. For example, the exponential distribution has a constant hazard rate λ. The Gompertz-Makeham distribution has an increasing failure rate to account for the increase in mortality, and its hazard rate is given by $\lambda(t) = \lambda + \alpha e^{\beta t}$. Since we model multiple failure rates and deadline-driven preemptions, our hazard rate also increases with time, and can be computed by simplifying $\lambda(t) = \frac{-r_1 e^{-r_1 t} - r_2 e^{r_2(t-b)}}{e^{-r_1 t} - e^{r_2(t-b)}}$.

After algebraic simplification, we get a more interpretable hazard rate for our model:

$$\lambda = r_2 + \bar{r}\left(\frac{1}{1 - e^{-r_2 b}e^{\bar{r}t}}\right) = \frac{r_1 + r_2 e^{-r_2 b}e^{\bar{r}t}}{1 - e^{-r_2 b}e^{\bar{r}t}} \quad (4)$$

Here, $\bar{r} = r_1 + r_2$, ie., the sum of the two failure rate constants.

For ease of exposition, we can write this as:

$$\lambda = \frac{r_1 + \gamma_1 e^{\delta(t-b)}}{1 - \gamma_2 e^{\delta(t-b)}} \quad (5)$$

We note that the numerator is similar to the hazard rate of Gompertz-Makeham. The key difference is the $1 - \gamma_2 e^{t-b}$ factor in the denominator. Recall that the sharp increase in preemption rate only happens close to the deadline, which means that $b \leq 24$. Thus, when $t < b$, we get a conventional $\lambda = r_1$, or the classic exponential distribution. As t approaches and exceeds b, the increase in failure rate kicks in, accounting for the deadline-driven rise in preemptions.

3.3 Connection to Statistical Mechanics

In addition to empirically-informed reliability models for preemptions, we also seek to understand *why* their distribution is bathtub shaped in order to provide a mechanistic understanding of preemptions and enhance the model interpretability and generalizability. To this end, we propose to "map" the problem of constrained preemptions to an equivalent problem of constrained many particle physical systems that can be analyzed employing the general theoretical framework of statistical mechanics.

A simple mapping can be established by assuming that the cluster management policy requires mutually exclusive preemptions. This assumption helps cluster management by serializing VM preemptions and making cluster operations easier, and it reduces the challenges associated with reacting to simultaneous preemptions. The preemption events become hard particles that cannot overlap. The probability distribution of N constrained preemption events gets mapped to the density of N randomly distributed particles confined in a 1-dimensional interval of length L (analogous to 24 hours). The particle size is equivalent to the length of the critical section, which is related to the preemption warning (30 seconds for Google PVMs). Finding the distribution of non-overlapping (i.e., hard) particles is a central problem in statistical mechanics [5]. This distribution is *not* uniform over the interval, but is higher towards

the ends of the interval—analogous to the bath-tub shaped nature of preemptions!

The analytical framework suggests that the bath-tub shape of the probability distribution is the key characteristic of constrained preemptions. Therefore, our approach will remain relevant in the face of future changes in preemption characteristics due to changes in cloud usage and operational policies.

4 APPLICATION POLICIES FOR CONSTRAINED PREEMPTIONS

For applications running on transient servers, the effects of preemptions can be ameliorated through different policies for fault tolerance and resource management. Prior work in transient computing has established the benefits of such policies for a broad range of applications. We now show how our empirical insights and analytical model can be used to develop policies for optimized execution of batch applications on Google Preemptible VMs.

4.1 Job Scheduling

Many cloud-based applications and services are *long-running*, and typically run a continuous sequence of tasks and jobs on cloud servers. In the case of deadline-constrained bathtub preemptions, applications face a choice: they can either run a new task on an already running server, or relinquish the server and run the task on a *new* server. This choice is important in the case of non-uniform failure rates, since the job's failure probability depends on the "age" of the server. Furthermore, since newly launched servers also have high preemption rates (and thus high job failure probability), the choice of running the job on an existing server vs. a new server is not obvious.

Our job scheduling policy uses the preemption model to determine the preemption probability of jobs of a given length T. Assume that the running server's age (time since launch) is s. Then, the the probability of failure on the existing server $P_{\text{Existing}} = max(1, F(T+s) - F(T))$. The alternative is to discard the server and launch a new server, in which case, the failure probability is $P_{\text{New}} = F(T)$. Depending on the server's age s and the job's running time T, we can compare P_{Existing} and P_{New}, and run the job on whichever case yields the lower failure probability.

4.2 Periodic Checkpointing

A common technique for reducing the total expected running time of jobs on transient servers is to use fault-tolerance techniques such as periodic checkpointing [**?**]. Checkpointing application state to stable storage (such as network file systems or centralized cloud storage) reduces the amount of *wasted work* due to preemptions. However, each checkpoint entails capturing, serializing, and writing application state to a disk, and increases the total running time of the application. Thus, the frequency of checkpointing can have a significant effect on the total expected running time.

Existing checkpointing systems for handling hardware failures in high performance computing, and for cloud transient servers such as EC2 spot instances, incorporate the classic Young-Daly periodic checkpointing interval that assumes that failures are exponentially distributed. That is, the application is checkpointed every $\tau = \sqrt{2 \cdot \delta \cdot \text{MTTF}}$ time units, where δ is the time overhead of writing a single checkpoint to disk.

However, checkpointing with a uniform period is sub-optimal in case of time dependent failure rates, and especially for bathtub failure rates. A sub-optimal checkpointing rate can lead to increased recomputation and wasted work, or result in excessive checkpointing overhead. Intuitively, the checkpointing rate should depend on the failure rate, and our analytical preemption model can be used for designing an optimized checkpointing schedule.

Let the uninterrupted running time of the job be T. Or in other words, T amount of work needs to be performed. Let the checkpoint cost be δ. We seek to minimize the total expected running time or the *makespan*, which is the sum of T, the expected periodic checkpointing cost, and the expected recomputation.

The makespan M can be recursively defined and computed. Let $M(W, t, i)$ denote the makespan when W length of job must be exectued, t is amount of job already done. We now need to determine when to take the *next* checkpoint, which we take after i steps. Let $E[M^*]$ denote the minimum expected makespan.

$$E[M^*(W,t)] = \min_{0<i\leq W} E[M(W,t,i)] \tag{6}$$

The makespan is affected by whether or not there is a failure *before* we take the checkpoint:

$$E[M(W,t,i)] = P_{\text{succ}}(t, i+\delta)\cdot E[M_{\text{succ}}] + P_{\text{fail}}(i+\delta, t)\cdot E[M_{\text{fail}}] \tag{7}$$

Here $P_{\text{succ}}(t, i+\delta)$ denotes the probability of the job successfully executing without failures until the checkpoint is taken, i.e., from t to $t+i+\delta$. $P_{\text{fail}}(t, i+\delta) = F(t+i+\delta) - F(i+\delta)$ is computed using the CDF, and $P_{\text{succ}} = 1 - P_{\text{fail}}$.

$E[M_{\text{succ}}]$ is the expected makespan when there are no job failures when the job is executing from step t to $t+i+\delta$, and is given by a recursive definition:

$$E[M_{\text{succ}}(W,t,i)] = t + i + \delta + E[M^*(W-i, t+i+\delta)] \tag{8}$$

Note that the makespan includes the amount of work already done $t+i$, the checkpointing overhead δ, and the expected minimum makespan of the rest of the job. Similarly, when the job fails before step i, then that portion is "lost work", and can be denoted by $E[L(t, i+\delta)]$ which denotes the expected lost work when there is a failure during the time interval t to $t+i+\delta$. A failure before the checkpoint means that we have made no progress, and W steps of the job still remain. The expected makespan in the failure case is then given by:

$$E[M_{\text{fail}}(W,t,i)] = E[L(t, i+\delta)] + E[M^*(W, t+i+\delta)] \tag{9}$$

In the case of memoryless distribution, $E[L(t, i+\delta)]$ is approximated as $\frac{i+\delta}{2}$. In our case, we use the failure CDF instead:

$$E[L(t, i+\delta)] = \int_t^{t+i+\delta} x f(x)\, dx, \tag{10}$$

where $f(x)$ is our probability density function represented by Equation 2.

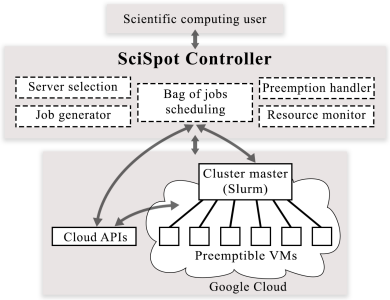

Figure 3: SciSpot architecture and system components.

Thus we can find the minimum makespan $E[M^*(W,t)]$ by using Equations 6–10. Given a job of length J, minimizing the total expected makespan involves computing $E[M^*(J,s)]$, where s is the current age of the server. Since the makespan is recursively defined, we can do this minimization using a dynamic programming approach, and extract the job-steps at which checkpointing results in a minimum expected makespan.

5 TESTING AND EVALUATION OF MODEL-INFORMED POLICIES

5.1 Details of the Experimental Framework used for Evaluation

SciSpot is a general-purpose software framework for running scientific computing applications on low-cost transient cloud servers. It incorporates policies and mechanisms for generating, deploying, orchestrating, and monitoring bags of jobs on cloud servers. Specifically, it runs a bag of jobs defined by these parameters:

Bag of job = {$\mathcal{A}$: Application to execute,
n: Number of jobs,
m: Minimum number of jobs to finish,
π: Generator function for job parameters,
$\mathcal{R}$: Computing resources per job}

SciSpot seeks to minimize the cost and running time of bags of jobs of scientific computing applications. SciSpot's cost and time minimizing policies for running bags of jobs are based on empirical and analytical models of the cost and preemption dynamics of transient cloud servers, which we present in the next section.

SciSpot is designed as a framework that increases the usability and viability of transient cloud servers for scientific computing applications, and provides a simple user interface to allow users to deploy their applications with minimum workflow changes. Most scientific computing applications are deployed on HPC clusters that have a batch scheduler such as Slurm [**?**] or Torque [**?**], and SciSpot integrates with these schedulers (e.g., Slurm) to provide the same interface to applications. As shown in Figure 3, SciSpot creates and manages clusters of transient cloud servers, manages all aspects of the VM lifecycle and costs, and implements the various policies described in the rest of this section.

High-level workflow: When a user wishes to run a bag of jobs, SciSpot handles the provisioning of a cluster of transient cloud servers. In addition, SciSpot deals with the scheduling and monitoring of the bag of jobs, and with VM preemptions. Execution of a bag of jobs proceeds in two phases. In the first phase, SciSpot selects the "right" cluster configuration for a given application through a cost-minimizing exploration-based search policy, described in Section **??**. In the second phase, SciSpot proceeds to run the remaining jobs in the bag on the optimal cluster configuration.

(Prateek: End design)

SciSpot is implemented as a light-weight, extensible framework that makes it convenient and cheap to run scientific computing applications in the cloud. We have implemented the SciSpot prototype in Python in about 2,000 lines of code, and currently support running VMs on the Google Cloud Platform [**?**]. SciSpot is implemented as a centralized controller, which implements the VM selection and job scheduling policies described in Section **??**. The controller can run on any machine (including the user's local machine, or inside a cloud VM), and exposes an HTTP API to end-users. Users submit bags of jobs to the controller via the HTTP API, which then launches and maintains a cluster of cloud VMs, and maintains the job queue and metadata in a local database. To improve usability, we automatically generate parameter combinations for a given bag size, based on a user-provided json file with ranges and constraints for each parameter.

SciSpot integrates, and interfaces with two primary services. First, it uses the Google cloud API [**?**] for launching, terminating, and monitoring VMs. Once a cluster is launched, it then configures a cluster manager such as Slurm [**?**] or Torque [**?**], to which it submits jobs. SciSpot uses the Slurm cluster manager, with each VM acting as a Slurm "cloud" node, which allows Slurm to gracefully handle VM preemptions. The Slurm master node runs on a small, 2 CPU non-preemptible VM, which is shared by all applications and users. SciSpot monitors job completions and failures (due to VM preemptions) through the use of Slurm call-backs, which issue HTTP requests back to the SciSpot controller.

In this section, we present empirical and analytical evaluation of the performance and cost of SciSpot with different scientific computing workloads and scales. Our evaluation consists of empirical analysis, as well as model-driven simulations for analyzing and comparing SciSpot behavior under different preemption and application dynamics.

We shed insight into the fundamental tradeoffs in constrained preemptions, the effectiveness of our model-based policies, and detailed empirical analysis of the cost and performance of our SciSpot framework with real-world scientific computing applications. We have already established the goodness of fit of our model and compared it to existing models earlier in Section **??**.

Environment and Workloads: All our empirical evaluation is conducted on the Google Public Cloud, and with these representative scientific computing applications:

Nanoconfinement. The nanoconfinement application launches molecular dynamics (MD) simulations of ions in nanoscale confinement created by material surfaces [**? ?**].

Shapes. The Shapes application runs an MD-based optimization dynamics to predict the optimal shape of deformable, charged nanoparticles [**? ?**].

LULESH. Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics (LULESH) is a popular benchmark for hydrodynamics simulations of continuum material models [**? ?**].

These examples are representative of typical scientific computing applications in the broad domain of physics, materials science,

and chemical engineering. These three examples are implemented as parallel programs that use OpenMP and MPI parallel computing techniques. The first two are used in nanoscale materials research [**? ? ? ? ? ?**] and LULESH is a widely used benchmark [**? ?**]. All applications are run with default parameters unless otherwise stated.

All applications use OpenMPI v2.1.1, are deployed on Slurm v0.4.3 and 64-bit Ubuntu 18.04, and run on Google Cloud VMs with x86-64 Intel Haswell CPUs. For completeness and to guard against concerns about poor cloud performance relative to HPC clusters [**? ? ? ? ?**], we benchmarked the Nanoconfinement application on the Big Red II cluster [**?**]. When run on 4 nodes with 16 CPUs each, the application takes 1140 seconds on Big Red II vs 850 seconds on SciSpot. We attribute the 20% improvement with SciSpot to the newer CPUs on Google Cloud (Intel Haswell vs. older 2012-era AMD Opterons in Big Red II).

5.2 Tradeoffs In Constrained Preemptions

Below, we analyze the effect of the bathtub shaped preemption curve on application running time.

Preemptions cause jobs to be restarted, which is the *wasted* time, which increases the total running time (i.e, makespan). In the case of constrained preemptions, we evaluate two preemption probability, the empirically observed bathtub shaped and a uniform distribution of failures.

In Figure 4 we show the expected wasted time *assuming the job suffers a single failure.* We see that in the case of uniform distribution, the increase is linear in the job length. Whereas with bathtub shaped distributions, the mirrors the shape of the CDF, and is constant for all but the largest of jobs.

The expected wasted time is also influenced by the probability of failure, and Figure 5 shows the expected waste. Because preemption rate is high in the initial stages for bathtub failures, for short jobs, the expected waste is higher, because they do not run long enough to take advantage of the low rate of preemptions in the middle stages. If preemptions were uniformly distributed, the waste is *quadratic*, and given by $\frac{T^2}{48}$, and is smaller for short jobs. We see that the "cross over" point is around 5 hours—and for longer jobs, a bathtub preemption behavior is preferable.

(Prateek: Do we give the equations here, or earlier?)

Result: *For constrained preemptions, bathtub distributions significantly reduce the expected increase in running times for medium to long running jobs, but are slightly inferior for short jobs.*

5.3 Model-based Policies

We now evaluate our model-driven policies.

5.3.1 Scheduling. Our job scheduling policy is model-driven and decides whether to request a new VM for a job or run on an existing VM.

Figure 6 shows the effect of this policy for a job of length 4 hours. We compare against a baseline of existing scheduling policies used by other systems such as ExoSphere . In the absence of insights about bathtub preemptions, the existing frameworks would continue to run on existing server. As the figure shows, for larger jobs the job failure probability is high.

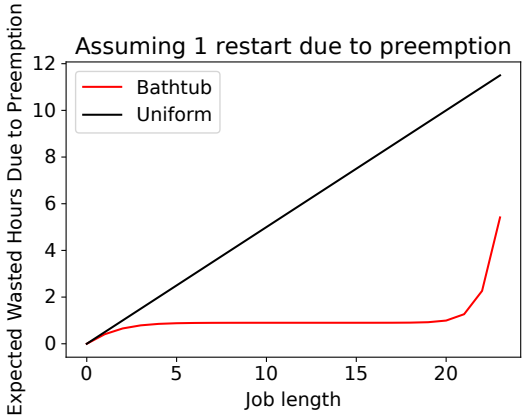

Figure 4: The expected wasted computation, given a single failure, is lower when preemptions are distributed in a bathtub shape, compared to uniformly distributed over the 24 hour interval

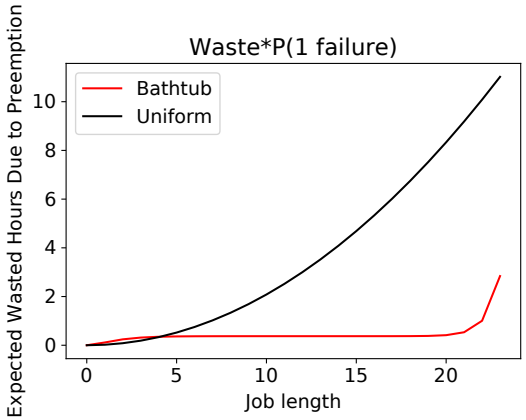

Figure 5: The expected wasted computation is lower when preemptions are distributed in a bathtub shape, compared to uniformly distributed over the 24 hour interval

Figure 7 shows the job failure probability for jobs of differen sizes for our model based policy vs a existing vanilla "null" policies that are not informed by the preemption dynamics.

5.3.2 Checkpointing.

5.4 Evaluation using Typical User-End Scientific Computing Applications

We show the empirical cost and performance and effectiveness of our SciSpot framework which incorporates our model-based insights and is an easy to use integrated Framework for scientific computing applications.

5.4.1 Cost. The primary motivation for using preemptible VMs is their significantly lower cost compared to conventional "on-demand" cloud VMs that are non-preemptible. Figure 11 compares

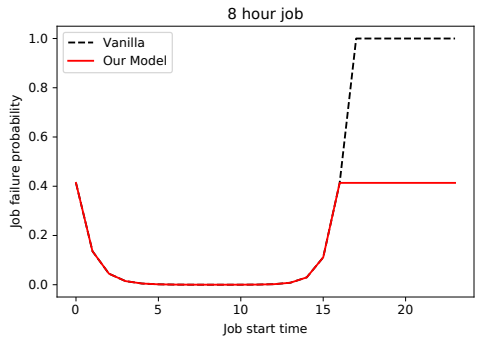

Figure 6: Job failure probability is reduced with the help of our deadline aware scheduling policy, especially for jobs starting near the end of the VM's lifetime

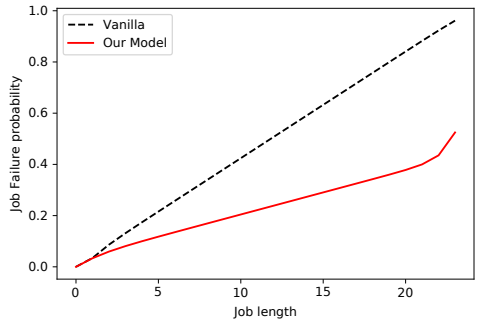

Figure 7: Job failure probability is lower with our deadline aware policy across all job sizes

Figure 8: Checkpointing overhead for jobs of length 4 at different starting times.

the cost of running different applications with different cloud VM deployments. SciSpot, which uses both cost-minimizing server selection, and preemptible VMs, results in significantly lower costs

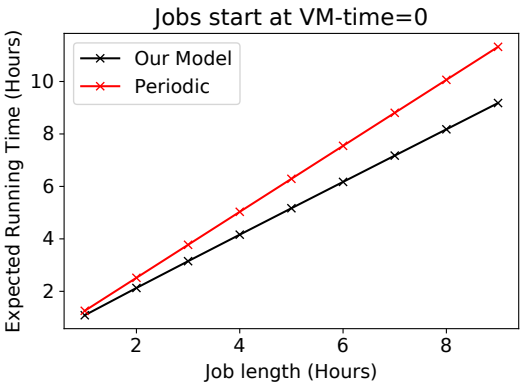

Figure 9: Running time with checkpointing when jobs start at time=0

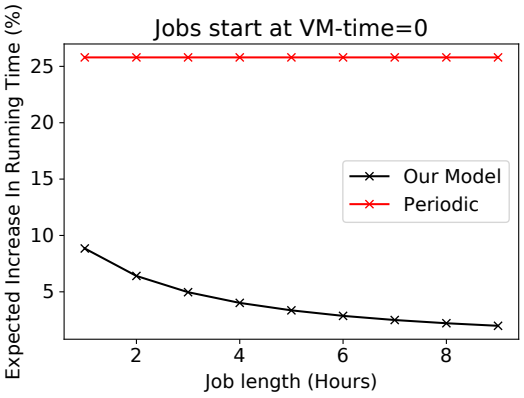

Figure 10: Increase in running time with Checkpointing when jobs start at time =0 (Should be an inset

.

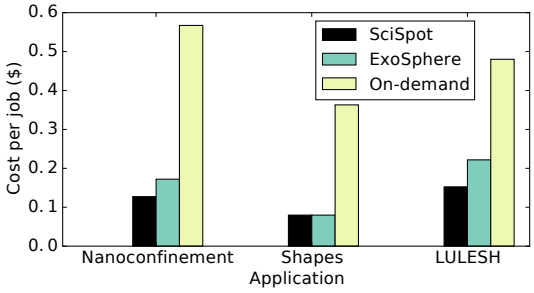

Figure 11: SciSpot's use of preemptible VMs can reduce costs by up to 5× compared to conventional cloud deployments, and 20% compared to the state of the art EC2 spot instance selection (ExoSphere [?]).

across the board, even when accounting for preemptions and recomputations. We also compare against ExoSphere [?], a state of the art system for transient server selection. ExoSphere implements a portfolio-theory approach using EC2 spot prices to balance average cost saving and risk of revocations using diversification and selecting VMs with low price correlation. However, this approach

is ineffective for the flat prices of Google Preemptible VMs. Unlike SciSpot, ExoSphere does *not* consider application performance when selecting servers, and thus is unable to select the best server for parallel applications. Since the Google `highcpu` VMs have the same price per CPU, ExoSphere picks an arbitrary "median" VM to break ties, which may not necessarily yield the lowest running times. This results in 20% cost increase over SciSpot.

Result: *SciSpot reduces computing costs by up to 5× compared to conventional on-demand cloud deployments.*

6 RELATED WORK

6.1 Transient Cloud Computing

The challenges posed by Amazon EC2 spot instances, the first transient cloud servers, have received significant attention from both academia and industry [**?**]. The significantly lower cost of spot instances makes them attractive for running preemption and delay tolerant batch jobs [**? ? ? ? ? ? ? ?**]. The distinguishing characteristic of EC2 spot instances is their dynamic auction-based pricing, and choosing the "right" bid price to minimize cost and performance degradation is the focus of much of the past work on transient computing [**? ? ? ? ? ? ? ? ? ? ? ?**]. However, as explained in Section **??**, it remains to be seen how Amazon's recent change [**?**] in the preemption model of spot instances affects prior work.

On the other hand, the effective use of transient resources provided by other cloud providers such as Google, Microsoft, Packet, and Alibaba largely remains unexplored. Ours is the first work that studies the preemption characteristics and addresses the challenges involved in running large-scale applications on the Google Preemptible VMs, and provides insights on the unique preemption dynamics.

6.1.1 Preemption Mitigation. Effective use of transient servers usually entails the use of fault-tolerance techniques such as checkpointing [**?**], migration [**?**], and replication. In the context of HPC workloads, [**? ? ?**] develop checkpointing and bidding strategies for MPI applications running on EC2 spot instances, based on older spot pricing models.

Periodic checkpointing [**?**] is not optimal in our case because preemptions are not memoryless. By treating bags of jobs as an execution unit, allowing some jobs to fail, and using insights from preemption models, we show that it is possible to reduce the recomputation times to acceptable levels even without the use of periodic checkpointing that imposes additional deployment and performance overheads. Our preemption model for Google preemptible VMs developed in Section 3 provides a novel characterization of bathtub shaped failure rates not captured by the classic Weibull distribution, and is distinct from prior efforts [**? ?**].

6.1.2 Server Selection. Optimized server selection is an important problem in cloud computing, and especially for transient servers because of the cost-performance-preemption tradeoff involved. Similar to SciSpot, SpotOn [**?**] is also a batch computing service that selects servers based on job characteristics and failure rates of different EC2 spot VMs. However, it is restricted to individual, single-VM batch jobs, and its design is tied to EC2 spot instances. The state of the art transient server selection involves the use of multiple types of VMs [**?**], and selecting a heterogeneous cluster can reduce the likelihood of mass concurrent preemptions. However, since scientific computing applications are mostly synchronous, even a single failure affects the entire job, and heterogeneous clusters are not required, and are in fact, detrimental [**?**]. Server selection is important even outside of preemptible VMs—developing bayesian optimization and application performance model based search for the "best" cloud VM is an active research area [**? ?**], but these techniques do not account for preemptions, which can substantially change the running time characteristics. Investigating these sophisticated search techniques for reducing exploration times is part of our future work.

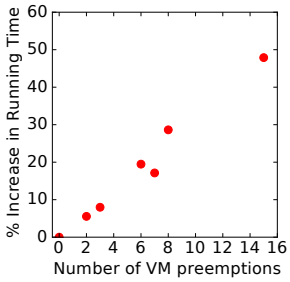

Figure 12: The increase in running time due to preemptions is under 50%, even at high preemption rates.

7 CONCLUSION

Given the rise of transient cloud computing and its use in web services and distributed data processing, it is not the question of if, but when, transient cloud computing becomes a credible and powerful alternative to HPC clusters for scientific computing applications. In this paper, we developed principled approaches for deploying and orchestrating scientific computing applications on the cloud, and presented SciSpot, a framework for low-cost scientific computing on transient cloud servers. SciSpot develops the first empirical and analytical preemption model of Google Preemptible VMs, and uses the model for mitigating preemptions for "bags of jobs". SciSpot's cost-minimizing server selection and job scheduling policies can reduce costs by up to 5× compared to conventional cloud deployments. When compared to HPC clusters, SciSpot can reduce the total job turnaround times by up to 10×.